

Федеральное государственное бюджетное образовательное учреждение высшего образования



«Московский государственный технический
университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ФУНДАМЕНТАЛЬНЫЕ НАУКИ
КАФЕДРА МАТЕМАТИКА И КОМПЬЮТЕРНЫЕ НАУКИ

Отчет

по лабораторному заданию № 16

Вариант 18

Дисциплина: Информатика

Название лабораторного задания: Использование рекурсии при программировании на C++.

Студент гр. ФН11-22Б

ХФ 26.04.21

(Подпись, дата)

М.Х. Хаписов
(И.О. Фамилия)

Преподаватель Доцент кафедры ИУ-6

Т.Н. Ничушкина

(Подпись, дата)

26.04.2021

Т.Н. Ничушкина
(И.О. Фамилия)

Москва, 2021

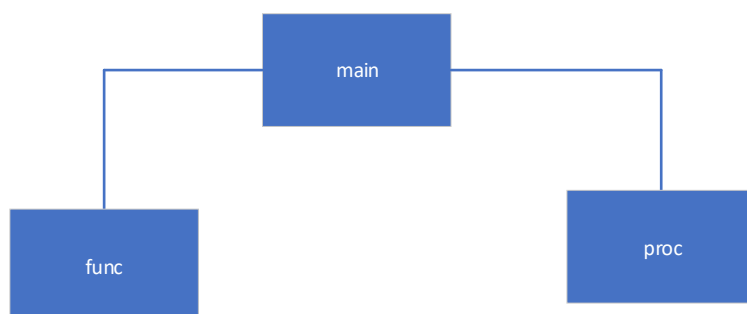
Цель: научиться использовать рекурсию при программировании на C++

Задача. С клавиатуры вводится символьная строка. Используя рекурсивный алгоритм, определить количество символов #,\$,@,*,&,% в заданной символьной строке. Реализовать программу с использованием двух видов подпрограмм – процедуры и функции. Вывести на экран всю необходимую информацию с соответствующими комментариями.

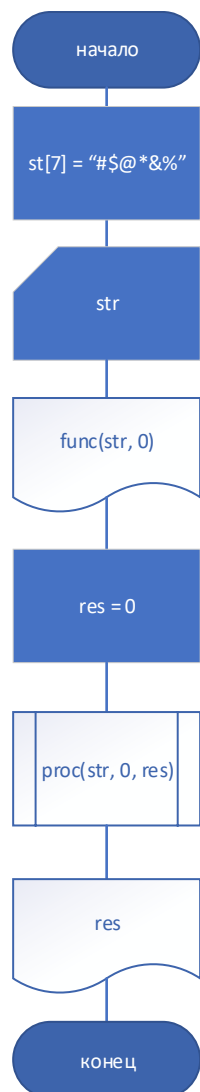
Базисное утверждение. Если индекс, с которого ведётся отчёт, превышает или становится равным длине строки, то количество заданных символов равно нулю.

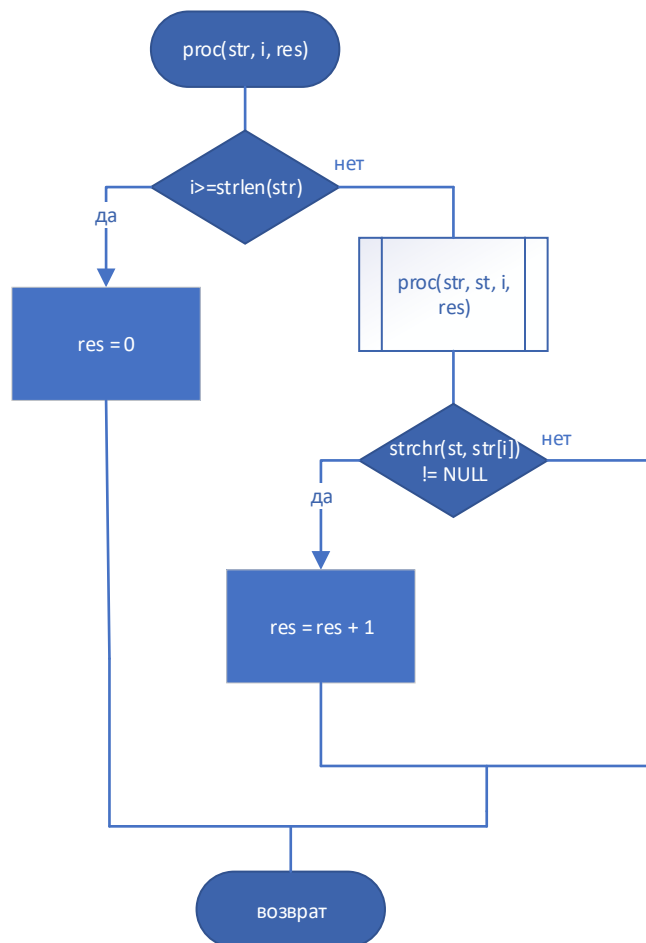
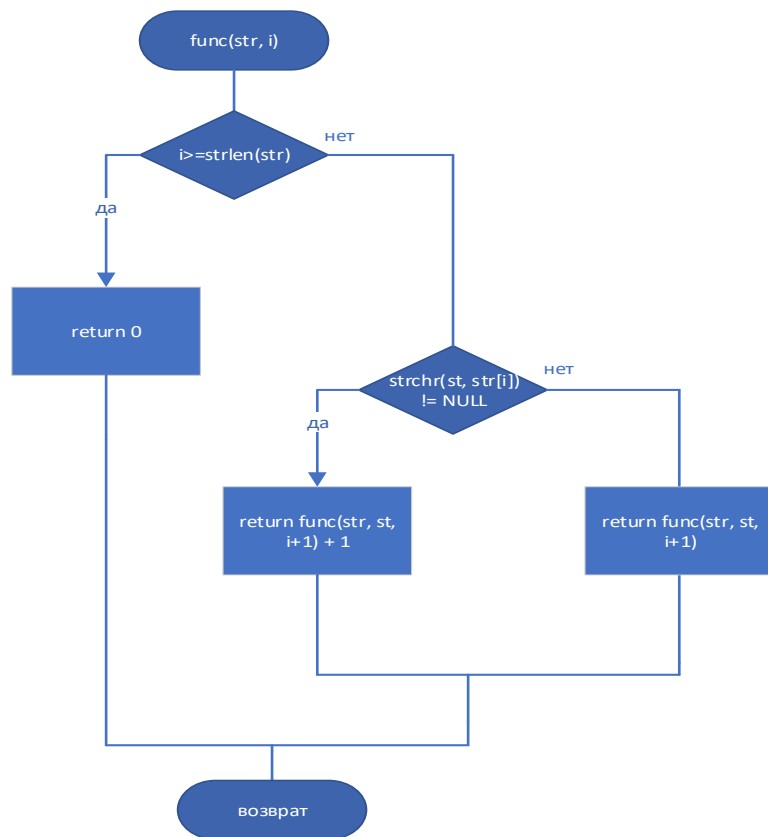
Рекурсивное утверждение. Если символ, находящийся в строке по заданному индексу, не является заданным, то количество заданных символов в строке, начиная с этого индекса, равно количеству заданных символов в этой же строке, начиная со следующего индекса. Если же символ, находящийся в строке по заданному индексу, всё же является заданным, то количество заданных символов в строке, начиная с этого индекса, ровно на единицу превышает количество заданных символов в этой же строке, начиная со следующего индекса.

Структурная схема



Схемы алгоритмов





Текст программы

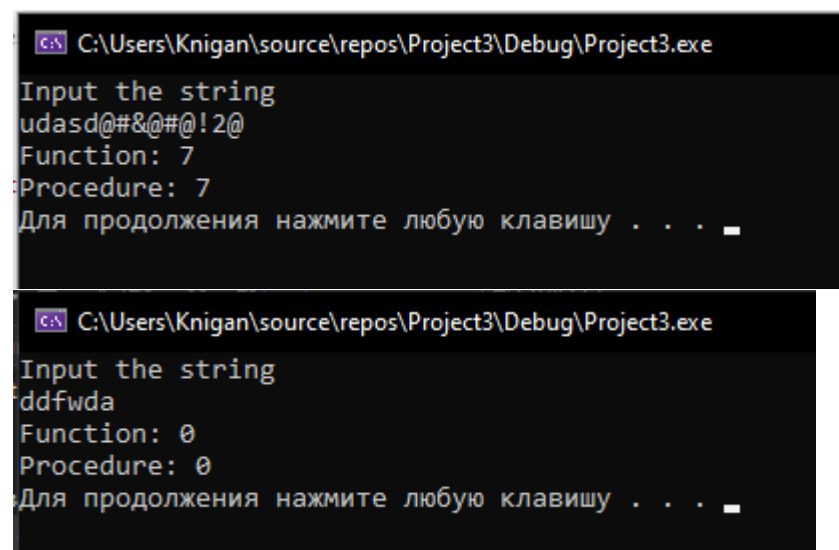
```
#include <iostream>

int func(char* str, char* st, int i) {
    if (i >= strlen(str)) return 0;
    else {
        if (strchr(st, *(str + i)) != NULL)
            return func(str, st, i + 1) + 1;
        else return func(str, st, i + 1);
    }
}

void proc(char* str, char* st, int i, int& res) {
    if (i >= strlen(str))
        res = 0;
    else {
        proc(str, st, i + 1, res);
        if (strchr(st, *(str + i)) != NULL)
            ++res;
    }
}

void main() {
    char st[7] = "$@*&";
    char* str = new char[64];
    std::cout << "Input the string\n";
    gets_s(str, 64);
    std::cout << "Function: " << func(str, st, 0) << std::endl;
    int res = 0;
    proc(str, st, 0, res);
    std::cout << "Procedure: " << res << std::endl;
    delete[] str;
    system("pause");
}
```

Тестирование



```
C:\Users\Knigan\source\repos\Project3\Debug\Project3.exe
Input the string
udasd@#&@#!2@
Function: 7
Procedure: 7
Для продолжения нажмите любую клавишу . . .

C:\Users\Knigan\source\repos\Project3\Debug\Project3.exe
Input the string
ddfwda
Function: 0
Procedure: 0
Для продолжения нажмите любую клавишу . . .
```

```
C:\Users\Knigan\source\repos\Project3\Debug\Project3.exe
Input the string
@@@###&&***
Function: 12
Procedure: 12
Для продолжения нажмите любую клавишу . . .

C:\Users\Knigan\source\repos\Project3\Debug\Project3.exe
Input the string
(5*5=25)&(5*8=40) = true
Function: 3
Procedure: 3
Для продолжения нажмите любую клавишу . . . _
```

Фрейм активации функции: 12 байт (служебная часть) + 12 байт (передаваемые параметры $4*3$) = 24 байт

Фрейм активации процедуры: 12 байт (служебная часть) + 16 байт (передаваемые параметры $4*4$) = 28 байт

Таким образом, использование функции для решения данной задачи является более эффективным и удобным, нежели использование процедуры, поскольку фрейм её активации меньше, чем у аналогичной процедуры.

Вывод: я научился использовать рекурсию при программировании на C++

Ответы на контрольные вопросы

1. Дайте определение рекурсии. Из каких двух частей состоит рекурсивное утверждение? (Ответ. рекурсия — определение, описание, изображение какого-либо объекта или процесса внутри самого этого объекта или процесса, то есть ситуация, когда объект является частью самого себя. Рекурсия состоит из базового и рекурсивного утверждений)

2. Что такое взаиморекурсия? Как программно ее можно реализовать?

(Ответ. *Косвенная* или *неявная* рекурсия образуется в случае цепочки вызовов других процедур, которые в конечном итоге приведут к вызову начальной)

```
void A(int j);
```

```
void B(int j)
```

```
{ ...
```

```
    A(j);
```

```
    ...
```

```
}
```

```
void A(int j)
```

```
{ ....
```

```
    B(j);
```

```
    ....
```

```
}
```

3. Что такое активация рекурсивной программы и фрейм активации? Как он влияет на емкостную сложность программы? Чем опасен большой объем фрейма активации? (Ответ. Каждое обращение к процедуре вызывает независимую *активацию* этой процедуры. С процедурой принято связывать некоторое множество локальных объектов (переменных, констант, типов и процедур), которые определены локально в этой процедуре, а вне ее не существуют или не имеют смысла. Совокупность данных, необходимых для одной активации называется *фреймом активации*)

4. Как рассчитать объем фрейма активации? Как можно уменьшить фрейм активации? (Если процедура обращается к себе несколько раз, образуется несколько одновременно существующих активаций и, соответственно, несколько фреймов активации. Обычно все фреймы являются различными экземплярами одной и той же структуры и размещаются в стеке. При некотором количестве вызовов возможно переполнение стека. На размер фрейма активации влияет способ передачи параметров в процедуру: при использовании параметров-

переменных фрейм содержит адреса данных (по 4 байта на каждый параметр), а при использовании параметров-значений - сами данные, которые могут занимать достаточно много места (например, массивы). Размер фрейма активации можно примерно определить следующим образом: $V = \langle \text{размер области параметров} \rangle + 4(\text{адрес возврата}) + 2 \cdot 8(\text{для сохранения содержимого регистров}) + \langle \text{размер области локальных переменных} \rangle + \langle \text{размер строки результата, если это функция, возвращающая результат - строку} \rangle$

5. Дайте описание структуры рекурсивной программы. (Ответ. Рекурсивная подпрограмма может быть реализована и как процедура и как функция. При реализации подпрограммы в виде процедуры в список параметров добавляется дополнительный параметр R, передаваемый по ссылке, через который процедура возвращает результат)

6. Какова особенность выполнения операторов до и после рекурсивного вызова? (Ответ. Когда функция вызывает сама себя, в стеке выделяется место для новых локальных переменных и параметров. Код функции работает с данными переменными. Рекурсивный вызов не создает новую копию функции. Новыми являются только аргументы. Поскольку каждая рекурсивно вызванная функция завершает работу, то старые локальные переменные и параметры удаляются из стека и выполнение продолжается с точки, в которой было обращение внутри этой же функции. Рекурсивные функции вкладываются одна в другую как элементы подзорной трубы)

7. Что такое линейная и древовидная рекурсия? Особенности спуска и подъема каждой из этих разновидностей рекурсий. (Ответ. Если подпрограмма вызывает сама себя только один раз в одной активации, то такая рекурсия называется линейной. Однако, есть рекурсивные определения, при вычислении по которым на каждой активации необходимо вычисление двух и более выражений, требующих дополнительного вызова рекурсивной подпрограммы. Такие рекурсии получили название древовидных)

8. Что такое полный и ограниченный перебор? (Ответ. Существует класс задач, в которых из некоторого количества вариантов необходимо выбрать наиболее подходящий (оптимальный). Для таких задач далеко не всегда удастся найти алгоритм, который позволил бы получить решение без анализа всех или большого количества комбинаций исходных данных, т.е. без осуществления перебора. Осуществление полного перебора требует много времени. Для решения задач данного типа применяют две стратегии: 1) формируют сразу всю комбинацию исходных данных и выполняют ее анализ в целом; 2) генерацию и анализ комбинации осуществляют по частям. Если имеется возможность, анализируя часть комбинации исходных данных, оценить ее перспективность с точки зрения получения решения, то вторая стратегия позволит получить результат за меньшее время за счет исключения из рассмотрения всех комбинаций, в которые входит данная часть)

9. Особенности задач полного перебора.

10. Способы уменьшения количества рассматриваемых вариантов при полном переборе. (Ответ. Существует две стратегии решения задач данного типа. При первой - мы генерируем вариант, а затем определяем его пригодность. При второй - вариант генерируется по одному элементу, и проверка пригодности осуществляется после добавления каждого элемента)

11. В чем преимущество использования рекурсии при решении задач полного и ограниченного перебора. (Ответ. В этом случае программный счетчик не поможет, так как он генерирует комбинации с повтором значений, и их нужно исключать из рассмотрения. Решение такой задачи упрощается при использовании древовидной рекурсии)

12. Дайте определение бинарного дерева. (Ответ. Бинарное дерево – это иерархическая структура данных, в которой каждый узел имеет не более двух потомков (детей). Как правило, первый называется родительским узлом, а дети называются левым и правым наследниками)

13. Чем отличается рекурсивная процедура построения бинарного дерева?

(Ответ. В математике бинарным (двоичным) деревом называют конечное множество вершин, которое либо пусто, либо состоит из корня и не более чем двух непересекающихся бинарных деревьев, называемых левым и правым поддеревьями данного корня. Таким образом, каждая вершина бинарного дерева может включать одно или два поддерева или не включать поддеревьев вовсе. Первое поддерево обычно называют левым, а второе – правым)

14. В чем особенности процедуры удаления вершины бинарного дерева?

(Ответ. При удалении вершины бинарного дерева возникают 3 ситуации: 1) узел не имеет потомков; 2) узел имеет одного потомка; 3) узел имеет двух потомков. В первом случае удаление производится без каких-либо иных действий. Во втором случае следует поставить потомка на место исходной вершины. В третьем случае, стоит либо пройти от вершины налево, а далее до упора вправо (пока не наткнёмся на лист), либо направо и до упора влево (пока не наткнёмся на лист). Найденный лист следует поставить на место исходной вершины)